The Microsoft Shared Source CLI Implementation

White Paper
June 2001

Standards are a key element in Microsoft Corp.'s commercial software business strategy. They provide a focal point for cooperation between Microsoft and a diverse group of participants: customers, industry partners, competitors, researchers and individuals. In October 2000, demonstrating a deep commitment to these communities, Microsoft and co-sponsors Hewlett-Packard Co. and Intel Corp. submitted specifications to ECMA, a well-known international standards association, for the core of the Microsoft® .NET Framework and the C# programming language. These specifications are currently undergoing the ECMA standardization process.

The ECMA standards documents will ultimately be the single most important reference for anyone planning to work with either C# or the language-independent runtime referred to as the Common Language Infrastructure, or CLI. Standards documents such as these should never be confused with implementation. The value of the precise language, of the community bonds formed by ongoing collaboration and of the deliberative discourse that accompanies their creation, is immense. Along with these valuable assets, Microsoft recognizes that freely available source code for a working implementation of such standards can be a useful adjunct tool for learning and experimentation.

With this in mind, Microsoft and Corel Corp. are currently working together on a freely available, shared source implementation of the C# and CLI working drafts. (More information on Microsoft's shared source philosophy is available at http://www.microsoft.com /sharedsource/.) This implementation is intended to meet the needs of academics, researchers, curious tinkerers and those who wish to build independent versions of the proposed ECMA standards. It will be a fully functional, illustrative implementation that can be used as a basis for noncommercial experimentation, as courseware, or as a guide for use by parties working on their own commercial ECMA implementations.

The shared source CLI implementation will contain source code for a complete implementation of the CLI, as well as two language compilers, one for the C# language and the other for the ECMAScript language. A beta-quality source code release is scheduled to be available later this year

for the Windows® platform and FreeBSD. The implementation is designed for use on PC hardware and shares much of its implementation with Microsoft's commercial CLI implementation, currently in beta format.

The ECMA CLI is an abstract program execution environment whose foundation is a virtual instruction set, or intermediate language, that is generated by compatible compilers or programming tools. Programs stored in this intermediate language (referred to as MSIL) are translated to native processor instructions using what is called an execution engine before being run. Since this translation step can be deferred until the last moment, it is often referred to as just-in-time (JIT) translation, although the name is a bit of a misnomer because it is often useful to perform the translation at earlier points in the program's life cycle for packaging or performance reasons. The shared source implementation includes JIT compilers for the i386 architecture.

The self-describing MSIL file format contains not only executable IL, but also metadata that capture salient characteristics of the IL. The file format is an extended form of the widely used PE/COFF executable format, and revolves around the fully described logical components, called assemblies, embedded within it. Besides containing the codes that implement their runtime behaviors, assemblies track references to other assemblies, support cryptographically strong naming, record version information and act as an enforced security boundary. Assemblies can be extended to include additional metadata through the use of custom attributes.

Compilers and development tools that target MSIL can choose to utilize the robust execution characteristics associated with garbage-collected memory management, exception-based error handling, and a complete virtualized type system that supports object-oriented mechanisms such as inheritance and interface-based object access in addition to the value types, by-reference types and pointers needed by implementers of procedural and functional languages. Because these mechanisms are implemented by the execution engine and abstracted away from the programming languages, it is possible not only to call between languages, but also to debug at source level, to inherit implementation and share data structures across languages.

MSIL is designed to be capable of expressing the semantics of many programming languages on many processor architectures. Programs can be written in such a way that faults caused by incorrect memory references during program execution won't damage other programs running, the execution engine or the operating system. Support for this technique, called type safe program verification, is included in the development tools and languages in this project, because it is a fundamental capability of the CLI execution engine.

In addition to type safety, the CLI supports a sophisticated resource control mechanism known as code access security. Traditional computer security mechanisms generally place access restrictions on data or other resources based on the notion of the current user's access rights and depend on operating system enforcement of these rights. Because the CLI manages access to all programming resources, access to these resources can be controlled not only by user identity but also by the level of trust placed in the running code itself. This approach enables security policies to be constructed based on information such as the source of an application, the presence of digital signatures or other evidence that a program should be trusted.

To be portable across operating systems and hardware architectures, the shared source implementation of the CLI is written to consume system resources through an API called the platform adaptation layer, or PAL. To peacefully coexist with native programs running on a host operating system, the PAL leverages the native operating system's threading model, file systems, networking capabilities, priority scheduling and synchronization mechanisms. If any of these functions are missing on a given platform, they can be provided within the PAL layer for that platform.

Philosophically, it is not the intent of the CLI to provide a restrictive least-common denominator for programmers; instead, the CLI is built to encourage programming languages to expose the unique aspects of each given platform to the programmers using them. A program that needs to access the underlying platform directly can do so by defining unmanaged interfaces in its metadata; these interfaces can either take advantage of standard CLI marshaling or implement their own custom marshaling schemes for communicating between code managed by the execution engine and code run externally in the host platform. Platform-specific calling conventions and data transformations also can be described in metadata and used from within the execution environment.

The ECMA working drafts specify class libraries that can act as a common runtime for any language built on the CLI. These class libraries, designed with platform neutrality in mind, include support for common programming tasks, support for the operating system abstractions found in the PAL, support for accessing the features of the execution engine, support for working with MSIL and limited support for application-level programming. Examples of the kind of runtime support that can be found in these class libraries include a wide variety of commonly needed structured data types, ways to use these data types in collections and ways to work with regular expressions. Operating system abstractions include support for networking, security, I/O and threading, while the execution

environment is exposed though classes targeting component reflection, garbage-collection control, platform-hosting interfaces, and unmanaged or nonstandard platform features. Finally, many of the common aspects of building Web services — such as object remoting and serialization, XML manipulation and security — are included.

As for the compilers included in the Microsoft Shared Source CLI Implementation, C# (pronounced “C sharp”) is a modern object-oriented, garbage-collected language that leverages much of the syntax of C while eliminating the programming constructs that sometimes lead to difficult-to-debug applications. Because C# was designed with the CLI, it uses the many runtime features found in the CLI to good effect. The C# compiler included will be an excellent reference for understanding how to use the mechanisms found in the CLI.

ECMAScript is a dynamic scripting language that, like C#, is object-oriented and
garbage-collected. It is often embedded into programs as an extension mechanism and has a number of interesting implementation choices that can be traced to this use. The ECMAScript compiler that is a part of this project is itself written in C#, and the dynamic aspects of the ECMAScript language show off different usage patterns of the CLI. More detailed information about the C# and ECMAScript languages is available in the ECMA working drafts and standards documents, Microsoft’s documentation, or one of the many excellent books on these subjects.

Industry analysts such as Gartner Inc. have identified Web services as the computing model that is clearly emerging for the purpose of spanning organizational and network boundaries. As programmers begin to deploy code in the form of Web services, they will demand the same

high levels of interoperability within that code as they already find within their XML-based communications components. C# and the CLI have been designed to anticipate these demands; they will be the most effective programming tools available for creating and delivering Web services. The Microsoft shared source CLI implementation will enable developers to understand these high-level interoperability mechanisms, regardless of the programming language or operating system choices that they have made, by providing an open, fully functional implementation. This implementation is being provided as a community resource for learning, for experimentation and for innovation around ECMA's standards work.

#########

Microsoft and Windows are either registered trademarks or trademarks of Microsoft Corp. in the United States and/or other countries.
The names of actual companies and products mentioned herein may be the trademarks of their respective owners.

For more information, press only:

John Hipsher, Waggener Edstrom, (503) 443-7000, johnh@wagged.com

Rapid Response Team, Waggener Edstrom, (503) 443-7070, rrt@wagged.com